



PROJEKT BASEL COIN

Modul 183



6. JANUAR 2024

GRUPPE: AIMO, AZIN, GIORGIO & ILIA

Abbildungsverzeichnis

Abbildung 1: Login Ansicht der BaselCoin.	3
Abbildung 2: Codeabschnitt vom Feature 1.	4
Abbildung 3: Codeabschnitt vom Feature 2.	5
Abbildung 4: Codeabschnitt vom Feature 4.	7

Versionsverzeichnis

Version	Datum	Änderung
1	27.02.2024	Projektdokumentation Erstelldatum
1.1	27.02.2024	Erste Einträge in die Dokumentation
1.3	01.03.2024	Letzter Eintrag in die Dokumentation

Inhalt

1	Einleitung	3
1.1	Was war der Auftrag?.....	3
2	Anwendung	3
2.1	Login.....	3
3	Sicherheitsfeatures der Basel Coin	4
3.1	Feature 1.....	4
3.2	Feature 2.....	5
3.3	Feature 3.....	6
3.4	Feature 4.....	7

1 Einleitung

1.1 Was war der Auftrag?

Unser Team wurde beauftragt, die Applikation "Basel Coin" zu erstellen, die zukünftig für die von Basel ausgegebene Kryptowährung zur Zahlung verwendet werden kann. Die Applikation muss folgende Sicherheitsmerkmale aufweisen:

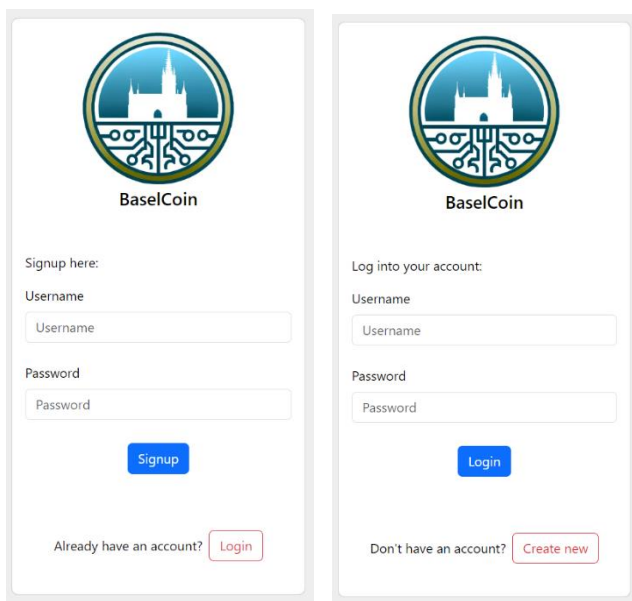
- Ein sicheres Login mit Schutz gegen Injection - Angriffe.
- Effektives Session - Management mit Timeout - Funktionen.
- Protokollierung von Ereignissen mit Datum, Uhrzeit, Benutzer und Aktion.
- Validierung aller Eingabefelder, wie Login und Passwort.

Der Admin kann neue Benutzer anlegen und Kontostände verwalten, während die Benutzer sich einloggen, ihren Kontostand sehen und sich abmelden können. Zusätzlich zum Projekt müssen wir eine kurze und prägnante Dokumentation zu den Sicherheitsmerkmalen erstellen, um das Projektteam der Stadt Basel über die Sicherheit zu informieren.

2 Anwendung

2.1 Login

Um die Anwendung nutzen und den Benutzer identifizieren zu können, haben wir einen Login-Bereich erstellt. Dort kann man sich entweder mit seinem bestehenden Konto verbinden oder sich für ein neues anmelden.:



The image displays two side-by-side screenshots of the BaselCoin application interface. Both panels feature the BaselCoin logo at the top, which consists of a blue circle containing a white silhouette of a city skyline with circuitry patterns below it. The left panel is titled 'Signup here:' and contains a 'Username' input field, a 'Password' input field, a blue 'Signup' button, and a link 'Already have an account? Login'. The right panel is titled 'Log into your account:' and contains a 'Username' input field, a 'Password' input field, a blue 'Login' button, and a link 'Don't have an account? Create new'.

Abbildung 1: Login Ansicht der BaselCoin.

3 Sicherheitsfeatures der Basel Coin

3.1 Feature 1

Login mit Schutz gegen Injection: Die Anwendung verfügt über ein sicheres Login-System, das Benutzern ermöglicht, sich mit Benutzername und Passwort anzumelden. Dabei wird darauf geachtet, dass die Eingabefelder ordnungsgemäss validiert werden, um Injection-Angriffe zu verhindern, wie zum Beispiel SQL-Injection oder Cross-Site Scripting.

```
// Beispiel für das sichere Aktualisieren von Daten mit Entity Framework Core
public async Task<IActionResult> UpdateUser(int id, [FromBody] User user)
{
    if (id != user.Id) return BadRequest();

    // EntityState.Modified setzt die Änderungen sicher um,
    // ohne anfällig für SQL-Injection zu sein.
    _context.Entry(user).State = EntityState.Modified;
    await _context.SaveChangesAsync();

    return NoContent();
}

// Beispiel für das sichere Löschen von Daten mit Entity Framework Core
public async Task<IActionResult> DeleteUser(int id)
{
    var user = await _context.Users.FindAsync(id);
    if (user == null) return NotFound();

    // Remove löscht die Daten sicher, geschützt vor SQL-Injection.
    _context.Users.Remove(user);
    await _context.SaveChangesAsync();

    return NoContent();
}

using Microsoft.EntityFrameworkCore;

// Beispiel für eine sichere Datenbankabfrage mit Entity Framework Core
public async Task<IActionResult> GetUserById(int id)
{
    // FindAsync verwendet parameterisierte Abfragen automatisch,
    // was SQL-Injection-Angriffe verhindert.
    var user = await _context.Users.FindAsync(id);
    if (user == null) return NotFound();
    return Ok(user);
}

// Beispiel für das sichere Hinzufügen von Daten mit Entity Framework Core
public async Task<IActionResult> CreateUser([FromBody] User newUser)
{
    // Add fügt die Daten sicher hinzu, ohne direkt SQL zu verwenden,
    // was vor SQL-Injection schützt.
    _context.Users.Add(newUser);
    await _context.SaveChangesAsync();
    return CreatedAtAction(nameof(GetUserById), new { id = newUser.Id }, newUser);
}
```

Abbildung 2: Codeabschnitt vom Feature 1.

3.2 Feature 2

Effektives Session-Management: Das Session-Management der Anwendung stellt sicher, dass Benutzersitzungen sicher verwaltet werden. Es gibt Mechanismen, um die Sitzung nach einer bestimmten Zeit der Inaktivität (Idle Timeout) oder nach einer festgelegten Zeit (absolute Timeout) automatisch zu beenden, um unbefugten Zugriff und Session-Hijacking zu verhindern.

```
using Microsoft.AspNetCore.Http;
using System.Threading.Tasks;

// Middleware zur Überwachung der Session-Timeouts
public class SessionTimeoutMiddleware
{
    private readonly RequestDelegate _next;
    private const int IdleTimeoutMinutes = 30; // Idle Timeout in Minuten
    private const int AbsoluteTimeoutMinutes = 120; // Absolutes Timeout in Minuten

    public SessionTimeoutMiddleware(RequestDelegate next)
    {
        _next = next;
    }

    public async Task InvokeAsync(HttpContext context)
    {
        var session = context.Session;
        var now = DateTimeOffset.UtcNow;

        // Überprüfung und Aktualisierung des letzten Zugriffszeitpunkts
        var lastAccess = session.GetString("LastAccess");
        var lastAccessTime = lastAccess != null ? DateTimeOffset.Parse(lastAccess) :

        // Prüfung auf Idle Timeout
        if (now - lastAccessTime > TimeSpan.FromMinutes(IdleTimeoutMinutes))
        {
            context.Session.Clear(); // Sitzung bei Idle Timeout löschen
            // Weiterleitung zur Login-Seite oder Anzeige einer Timeout-Nachricht
            context.Response.Redirect("/login?timeout=idle");
            return;
        }

        // Prüfung auf absolutes Timeout
        var sessionStart = session.GetString("SessionStart");
        var sessionStartTime = sessionStart != null ? DateTimeOffset.Parse(sessionSta

        if (now - sessionStartTime > TimeSpan.FromMinutes(AbsoluteTimeoutMinutes))
        {
            context.Session.Clear(); // Sitzung bei absolutem Timeout löschen
            // Weiterleitung zur Login-Seite oder Anzeige einer Timeout-Nachricht
            context.Response.Redirect("/login?timeout=absolute");
            return;
        }

        // Aktualisierung des letzten Zugriffszeitpunkts
        session.SetString("LastAccess", DateTimeOffset.UtcNow.ToString());

        await _next(context);
    }
}

// Hinzufügen der Middleware in der Startup-Klasse
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    app.UseSession(); // Stellt sicher, dass die Session-Middleware zuerst ausgeführt
    app.UseMiddleware<SessionTimeoutMiddleware>(); // Hinzufügen der Session-Timeout-

    // Weitere Konfigurationen...
}
```

Abbildung 3: Codeabschnitt vom Feature 2.

3.3 Feature 3

Protokollierung von Ereignissen: Die Anwendung protokolliert alle relevanten Ereignisse, wie beispielsweise Benutzeranmeldungen, Transaktionen oder Änderungen an Benutzerkonten. Jeder Eintrag im Applikationslog enthält Datum, Uhrzeit, den Typ des Ereignisses, den betroffenen Benutzer und die durchgeführte Aktion. Dadurch wird eine lückenlose Nachverfolgung von Aktivitäten ermöglicht, was die Sicherheit und Nachvollziehbarkeit der Anwendung verbessert.

3.4 Feature 4

Input-Validierung: Alle Eingabefelder der Anwendung, einschliesslich Login, Passwort und andere Formularfelder, werden sorgfältig validiert, um sicherzustellen, dass nur gültige und erwartete Eingaben akzeptiert werden. Dadurch wird das Risiko von Angriffen, die auf fehlerhafte oder bösartige Eingaben abzielen, minimiert.

```
using System.ComponentModel.DataAnnotations;

// Beispiel für ein Benutzermodell mit Validierungsattributen
public class User
{
    [Required(ErrorMessage = "Benutzername ist erforderlich.")]
    [StringLength(50, MinimumLength = 3, ErrorMessage = "Benutzername muss zwischen 3 und 50 Zeichen liegen.")]
    public string Username { get; set; }

    [Required(ErrorMessage = "Passwort ist erforderlich.")]
    [StringLength(100, MinimumLength = 6, ErrorMessage = "Passwort muss zwischen 6 und 100 Zeichen liegen.")]
    [DataType(DataType.Password)]
    public string Password { get; set; }

    // Beispiel für weitere Felder mit spezifischer Validierung
    [EmailAddress(ErrorMessage = "Ungültige E-Mail-Adresse.")]
    public string Email { get; set; }

    [Range(0, 100, ErrorMessage = "Wert muss zwischen 0 und 100 liegen.")]
    public int Age { get; set; }
}

// Beispiel für eine Controller-Methode, die die Validierung nutzt
[HttpPost]
public IActionResult CreateUser([FromBody] User newUser)
{
    if (!ModelState.IsValid)
    {
        // Rückgabe einer Fehlermeldung, wenn die Validierung fehlschlägt
        return BadRequest(ModelState);
    }

    // Logik zum Speichern des neuen Benutzers, wenn die Validierung erfolgreich war
    // (Implementierungsdetails ausgelassen)

    return Ok();
}
```

Abbildung 4: Codeabschnitt vom Feature 4.